Unit 4: Localization Part 4
Monte Carlo Localization

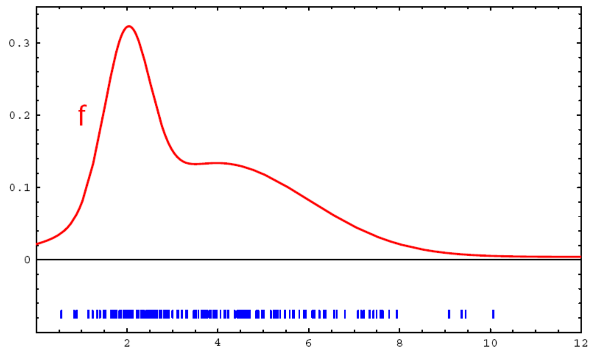
Computer Science 4766/6912

Department of Computer Science
Memorial University of Newfoundland

June 13, 2018

- Representing a probability distribution in a grid has inherent problems:
  - Coarse-grained grids inaccurate
  - Fine-grained grids expensive
- Another popular way of representing a probability distribution is to assume it is a particular distribution (e.g. Gaussian or mixture of Gaussians → Kalman filter)
- In Monte Carlo localization we represent a probability distribution with a set of samples drawn from that distribution
- The estimation of a sampled representation is known by other names: particle filters, condensation, "survival of the fittest"
- For a derivation of the algorithm see [Thrun et al., 2005]

Consider the distribution f given below; A particle filter represents this distribution by a set of samples randomly drawn from the distribution



The samples shown on the bottom are also known as **particles**

A particle filter is just the application of Bayes filter to estimate a probability distribution, where that distribution is represented by a set of samples

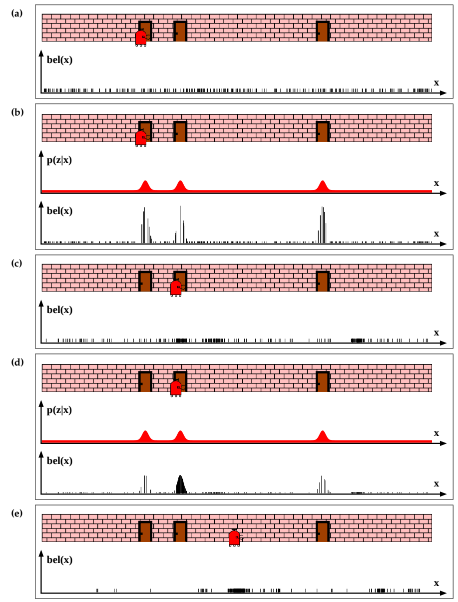


Figure 8.11 Monte Carlo Localization, a particle filter applied to mobile robot localization.

Monte Carlo localization has the two usual steps of Bayes filter: prediction and measurement update; We will consider prediction first

First some notation:

$x_t^{[m]}$ The m^{th} particle, which represents one guess about the state x_t ($x_t^{[m]}$ has the same dimensionality as x_t)

$\mathcal{X}_t$ The set of particles:

$$\mathcal{X}_t = \left\{ x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]} \right\}$$

where M is the number of particles

Prediction

1. Particle_Filter_Prediction($\mathcal{X}_{t-1}, u_t$)
2. $\overline{\mathcal{X}}_t = \emptyset$
3. for $m = 1$ to M do
4. sample $x_t^{[m]} \sim p(x_t | u_t, x_{t-1}^{[m]})$
5. $\overline{\mathcal{X}}_t = \overline{\mathcal{X}}_t + x_t^{[m]}$
6. endfor
7. return $\overline{\mathcal{X}}_t$

The set of particles $\overline{\mathcal{X}}_t$ represents $\overline{bel}(x_t)$; But how does this 'sample' step work?

Consider again the odometry motion model:

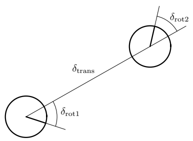


Figure 5.7 Odometry model: The robot motion in the time interval $(t-1, t]$ is approximated by a rotation δ_{rot1}, followed by a translation δ_{trans} and a second rotation δ_{rot2}. The turns and translations are noisy.

Previously, we obtained the δ parameters from the robot's movement u_t; We then obtained the $\hat{\delta}$ parameters we would expect for a *possible* movement from x_{t-1} to x_t; Our goal was to estimate $p(x_t | u_t, x_{t-1})$

Now we wish to sample from $p(x_t | u_t, x_{t-1})$; That is, given a particular u_t and x_{t-1} we want to obtain one possible value for x_t, drawn at random with probability $p(x_t | u_t, x_{t-1})$

Method: We obtain the δ parameters from u_t (as before); We then add random noise to these to generate the $\hat{\delta}$ parameters ; Finally, we compute x_t as a movement from x_{t-1} , specified by the $\hat{\delta}$ parameters

1. sample_motion_model_odometry(u_t, x_{t-1})
2. $\delta_{rot1} = \mathrm{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$
3. $\delta_{trans} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$
4. $\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1}$
5. $\hat{\delta}_{rot1} = \delta_{rot1} - \mathrm{sample}(\sigma^2_{rot1})$
6. $\hat{\delta}_{trans} = \delta_{trans} - \mathrm{sample}(\sigma^2_{trans})$
7. $\hat{\delta}_{rot2} = \delta_{rot2} - \mathrm{sample}(\sigma^2_{rot2})$
8. $x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1})$
9. $y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1})$
10. $\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$
11. return $x_t = (x', y', \theta')^T$

where $\mathrm{sample}(\sigma^2)$ draws a random value from a Gaussian distribution with variance σ^2

Shown below are 500 samples obtained from sample_motion_model_odometry using three different sets of *alpha* parameters,

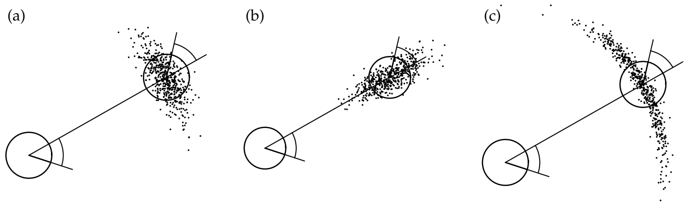


Figure 5.9 Sampling from the odometry motion model, using the same parameters as in Figure 5.8. Each diagram shows 500 samples.

The algorithm Particle_Filter_Prediction uses sampling to produce the new set of particles $\overline{X}_t$ from the old set X_{t-1}

In the limit as $M \to \infty$ the distribution of $\overline{X}_t$ will approximate $\overline{bel}(x_t)$; For finite M this is only an approximation

To obtain $bel(x_t)$ we must take measurements into account; This is accomplished by giving each particle, $x_t^{[m]}$, a weight of $p(z_t|x_t^{[m]})$; We then randomly select the new particle set X_t from $\overline{X}_t$ by picking M particles at random with the probability of each particle being selected as proportional to its weight $w_t^{[m]}$

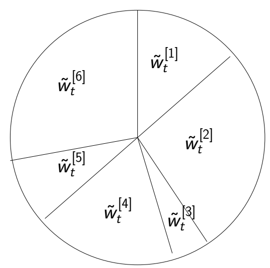
Incorporating this step yields the full particle filter algorithm...

Particle Filter Algorithm

1. Particle_Filter(X_{t-1}, u_t, z_t)
2. $\overline{X}_t = X_t = \emptyset$
3. for $m = 1$ to M do
4. sample $x_t^{[m]} \sim p(x_t|u_t, x_{t-1}^{[m]})$
5. $w_t^{[m]} = p(z_t|x_t^{[m]})$
6. $\overline{X}_t = \overline{X}_t + \left\langle x_t^{[m]}, w_t^{[m]} \right\rangle$
7. endfor
8. for $m = 1$ to M do
9. draw i with probability $\propto w_t^{[i]}$
10. add $x_t^{[i]}$ to X_t
11. endfor
12. return X_t

The weight $w_t^{[m]}$ gives each particle its "chance of survival" into the next particle set; Particles with high weight represent robot poses which appear similar to the robot's current sensory observation

How do we "draw i with probability $\propto w_t^{[i]}$"; Here i is the i^{th} particle; The easiest method is known as **roulette wheel selection**; Imagine a roulette wheel with the size of different segments of the wheel set by the weights,



A particular rotation angle of the wheel is then selected with uniform probability; The probability that the wheel's angle falls in slice i is given by $\tilde{w}_t^{[i]}$; How does this relate to $w_t^{[i]}$?

$$\tilde{w}_t^{[i]} = \frac{w_t^{[i]}}{s}, \qquad \text{where } s = \sum_j w_t^{[j]}$$

The second for loop in the particle filter implements **resampling**:

- for $m = 1$ to M do
- draw i with probability $\propto w_t^{[i]}$
- add $x_t^{[i]}$ to X_t
- endfor

From the particles in the prediction step, $\overline{X}_t$, we select M particles to go into X_t; Often we will get many copies of the same particle carrying over into X_t; Thus, resampling reduces diversity and focusses particles on areas of the state space where $bel(x_t)$ is large

Examples:

- Revisit figure 8.11
- video: particleFilter.avi

Issues of MCL:

- **Particle deprivation** occurs when there are no particles in the vicinity of the robot's true pose
- The prediction step increases diversity, while resampling reduces it; It may be advantageous to reduce the frequency of resampling to prevent particle deprivation
- If M is large particle deprivation is less likely (still possible) but the computational expense may be high; If M is small then particle deprivation is more likely, but the computational expense will be lower
- Particle deprivation can also be addressed by the addition of random particles; This has the added benefit of solving the kidnapped robot problem (if one of the random particles is close enough to the location of the kidnapped robot)

Particle filters are very popular. Why?

- The particles in MCL can accomodate complex multi-modal probabilty distributions, thus supporting multi-hypothesis belief representation
- The fact that the motion and measurement models represent non-linear functions is not problematic (as it is for the Kalman filter)
- We can tradeoff computational complexity for accuracy by varying the number of particles
- They are (relatively) easy to implement!

References

Thrun, S., Burgard, W., and Fox, D. (2005).
Probabilistic Robotics.
MIT Press.